Sensor Task Manager (STM)

V.S. Subrahmanian

University of Maryland

Joint work with:
F. Ozcan, IBM Almaden

T.J. Rogers, University of Maryland

Scaling task handling

- Users specify tasks of interest:
  - Where to monitor
  - When to monitor
  - Monitoring conditions to check for
  - What to do when monitoring conditions arise.
- Data on the ground changes continuously.
- Monitoring conditions need to be evaluated continuously.
- LOTS of conditions, LOTS of sensed data. Scalability is key.

How to Handle lots of tasks

- Three pronged strategy:
  - **Merge:** Merge tasks to eliminate any redundancy using a cost model. Such merging only works well for relatively small sets of tasks (or conditions to evaluate).
  - **Task Assignment:** Select sensors (and/or data sources) to handle merged tasks so as to optimize performance criteria.
  - **Partition:** Given a large set of tasks (or conditions) to process, determine ways of partitioning into smaller sets of manageable size.
- For time reasons, only the last is discussed today.

Task Partitioning

- Goal: Partition large number of tasks into disjoint sets and minimize the total cost of executing the tasks
- Cost estimation function (cost): approximates the cost of executing a set of tasks together. Any function satisfying the axioms:
  - $T_i \subseteq T_j \rightarrow \text{cost}(T_i) \leq \text{cost}(T_j)$
  - $\text{cost}(\emptyset) = 0$

Partitions

- Partition: A partition P of a set T of tasks is a set $\{ P_1,\ldots,P_n \}$, where each P_i is non-empty, $i \neq j \rightarrow P_i \cap P_j = \varnothing$ and

$$\bigcup_{q_i \in P_j, j=1,..n} P_j = T$$

- Each P_i is called a component of P.
- P is a sub-partition of Q if $\bigcup_{q_i \in P_j, j=1,..n} P_j \subset T$

Task Partitioning Problem (TP)

- Formal Problem Definition: Given as input a set T of tasks, and a cost estimation function *cost* , find a partition $P = \{ P_1, \ldots, P_n \}$ such that $\sum_{i=1}^{n} \text{cost}(P_i)$ is minimized

- Need to balance execution time of tasks vs. optimization time of tasks.

TP Algorithms

- Theorem: The task partitioning problem is NP-complete.
- Proposed multiple types of algorithms to solve TP
  - A*-based : Finds optimal solution
  - Branch-and-Bound (BAB): Finds optimal solution
  - Greedy: Is not guaranteed to find optimal solution, has polynomial running time – several variants proposed.

Adaptation of the A* Algorithm

- State: A *sub-partition* of T;
  $P' = \{P_1,\ldots,P_m\}$
  - Start state: Empty partition
  - Goal state: A *partition* of T
- Ex: $T = \{t_1, t_2, t_3, t_4, t_5\}$
  - Example state $s = \{\{t_1, t_3\}, \{t_2, t_5\}\}$
  - Goal State $= \{\{t_1, t_3, t_4\}, \{t_2, t_5\}\}$
- $g(s) = \Sigma_{P_i \in P'} \text{cost}(P_i)$
- Ex: $g(s) = \text{cost}(\{t_1, t_3\}) + \text{cost}(\{t_2, t_5\})$

Adaptation of the A* Algorithm

- Expansion function
  - Pick a task t and insert it into each component P_i of P'
  - Create a new component P_{m+1} containing only t
- Ex : $\{\{t_1\}, \{t_2\}\}$ and we pick t_4, then
  - $\{\{t_1,t_4\}, \{t_2\}\}$,
  - $\{\{t_1\}, \{t_2, qt_4\}\}$
  - $\{\{t_1\}, \{t_2\}, \{t_4\}\}$,

Adaptation of the A* Algorithm

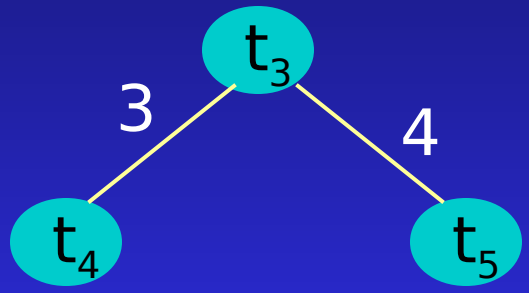
- $h(s) = \min\{incr(t,s) \mid t \notin P\}$
  - $incr(t,s) = \min\{cost(t), \min\{cost(t \cup P_i) - cost(P_i) \mid P_i \in P'\}\}$
- Ex: $s = \{\{t_1\}, \{t_2, t_5\}\}$
  - $h(s) = \min\{incr(t_3,s), incr(t_4, s)\}$
  - $incr(t_4, s) = \min\{cost(t_4), (cost(\{t_1, t_4\}) - cost(t_1)), (cost(\{t_2, t_5, t_4\}) - cost(\{t_2, t_5\}))\}$
- Theorem: The function *h* is admissible and satisfies the monotone restriction.
- Theorem: hence, A* finds an optimal partition.

Cluster Graphs

- Canonical Cluster Graph (T): Undirected weighted graph where
  - $V = \{ \{t_i\} \mid t_i \in T \}$
  - $E = \{ (\{t_i\},\{t_j\}) \mid t_i, t_j \in T \text{ and } w(\{t_i\},\{t_j\}) > 0$
  - $w(\{t_i\},\{t_j\}) = cost(t_i) + cost(t_j) - cost(\{t_i,t_j\})$

Cluster Graph Example

- $T = \{t_1, t_2, t_3, t_4, t_5\}$
- $cost(t_i) = 5$, $cost(\{t_1, t_2\}) = 8$, $cost(\{t_3, t_4\}) = 7$ and $cost(\{t_3, t_5\}) = 6$



Greedy Partitioning Algorithm

- Builds the partition iteratively using a cluster graph representation
- In each iteration, finds the edge (t_i,t_j) with the maximum weight and removes from the graph
- Terminates when all edges are processed
- Running time : O(|V|.|E|)

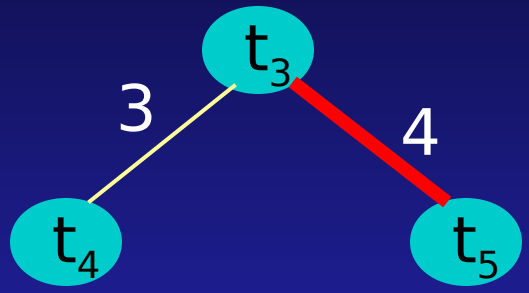
Greedy Partitioning Algorithm

- At each step, four possible cases
  - *Case 1:* Both t_i and t_j are in the same component; do nothing
  - *Case 2:* One of t_i or t_j is in a component; insert the other one into the same component
  - *Case 3:* Neither is in any of the components; create a new component with t_i and t_j
  - *Case 4:* t_i and t_j are in different components; move one of them into the other component, or leave as it is

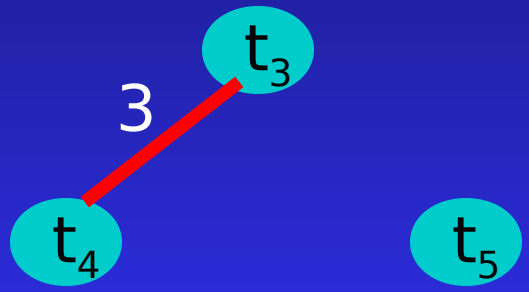
Running Example

$T = \{t_1, t_2, t_3, t_4, t_5\}$

$P = \{\}$

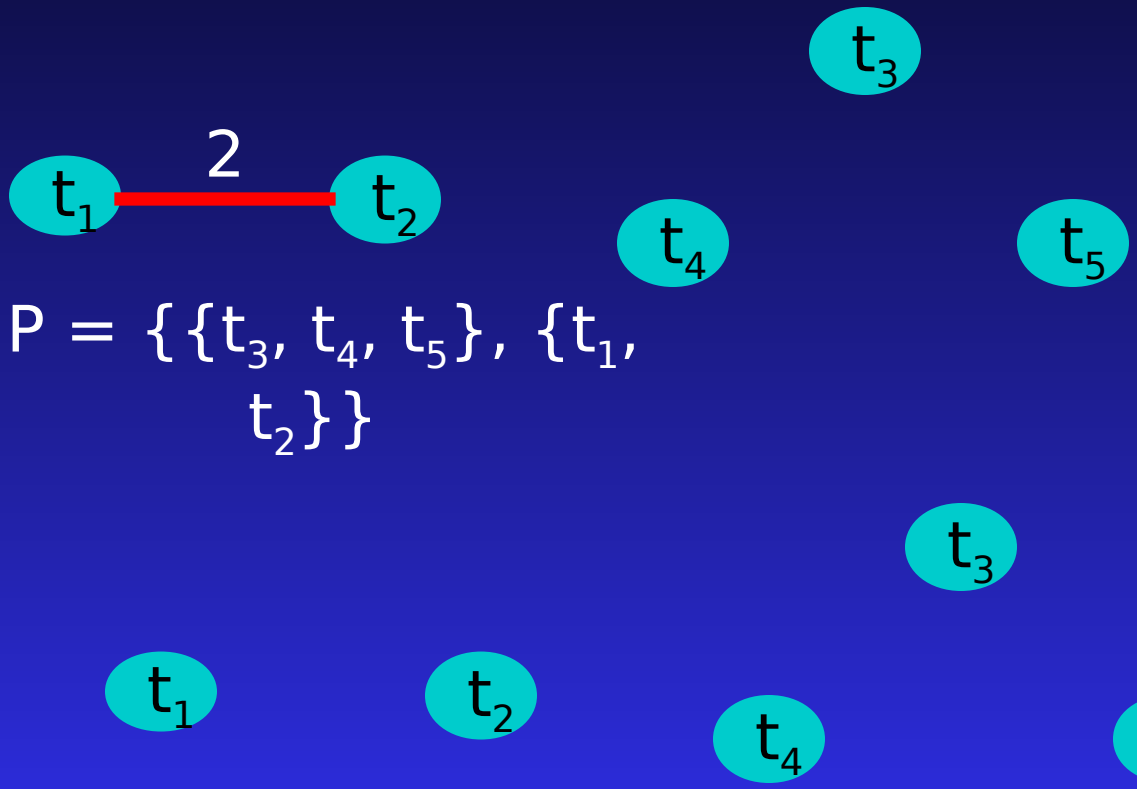


$P = \{\{t_3, t_5\}\}$



Running Example, cont.

$P = \{\{t_3, t_4, t_5\}\}$



$P = \{\{t_3, t_4, t_5\}, \{t_1, t_2\}\}$

Variants of the Greedy Algorithm

- Several variants of the basic greedy algorithm (5 in all we worked with, 2 examples below)
  - Greedy with weight update (Greedy w/ WU)
    - After inserting tasks into components, it updates the weights of adjacent edges
  - Greedy with no move around (Greedy w/ NMA)
    - Once a task is inserted into a component, it stays there.

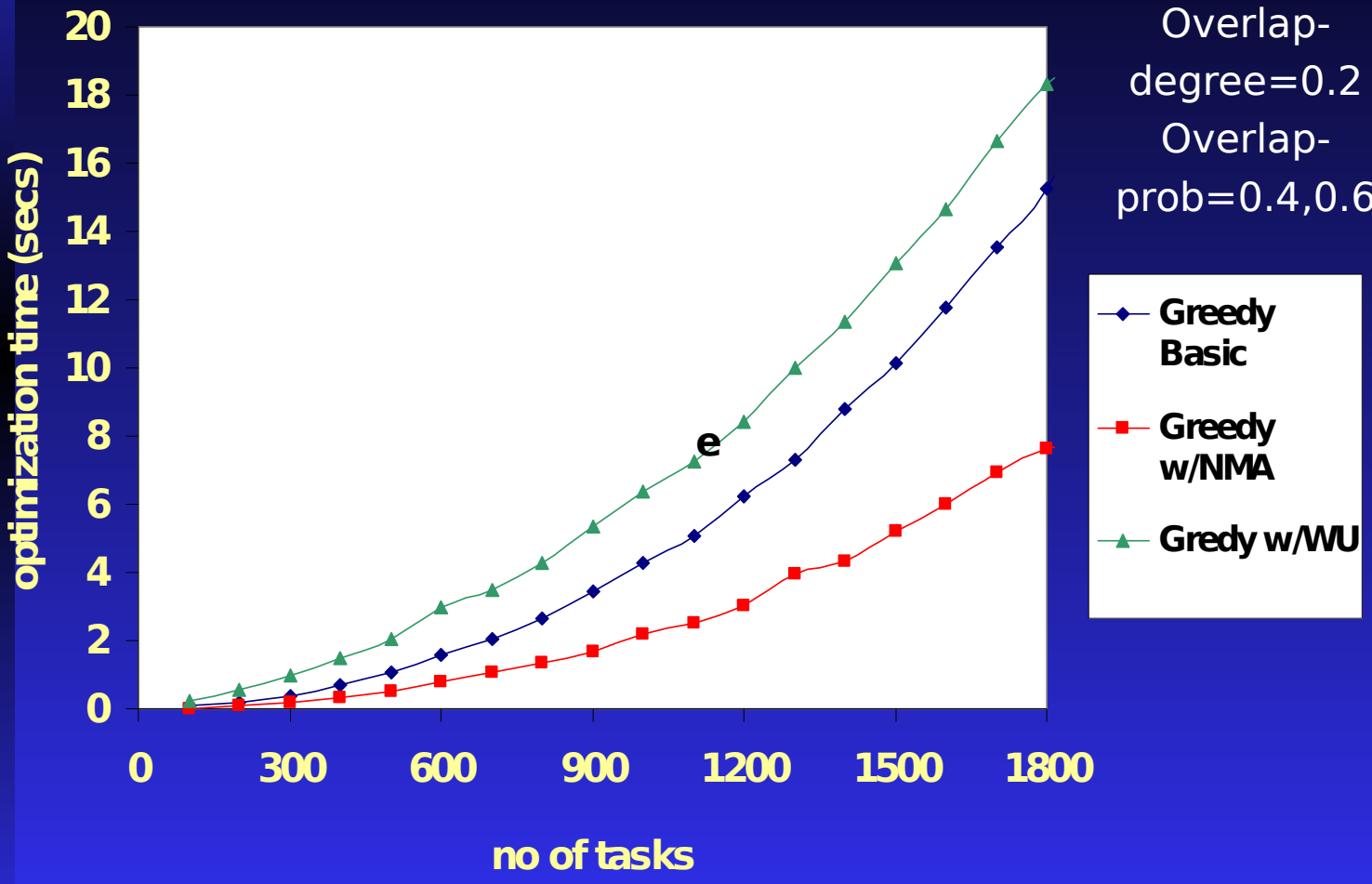
Running Times

No of tasks	A*-based	BAB	Greedy Basic	Greedy w/ NMA	Greedy w/ WU
5	68.3	81.1	2.2	0.3	1
6	328.5	417.1	2.75	0.35	1.55
7	1420.7	1570.5	2.8	0.4	1.7
8	9941.2	6752	3.3	0.45	3
9	18281.9	5243.4	4.7	0.5	4.8
10	44782.6	10109	6	0.5	6.6

Execution times (millisecs) (Cost-limit = 100, Overlap-degree=0.6, Overlap-prob = 0.4,0.6)

Only 10 tasks above as A* runs out of space. BAB can do 11 or 12. Greedy methods can handle thousands (see n

Scalability of The Greedy Algorithms

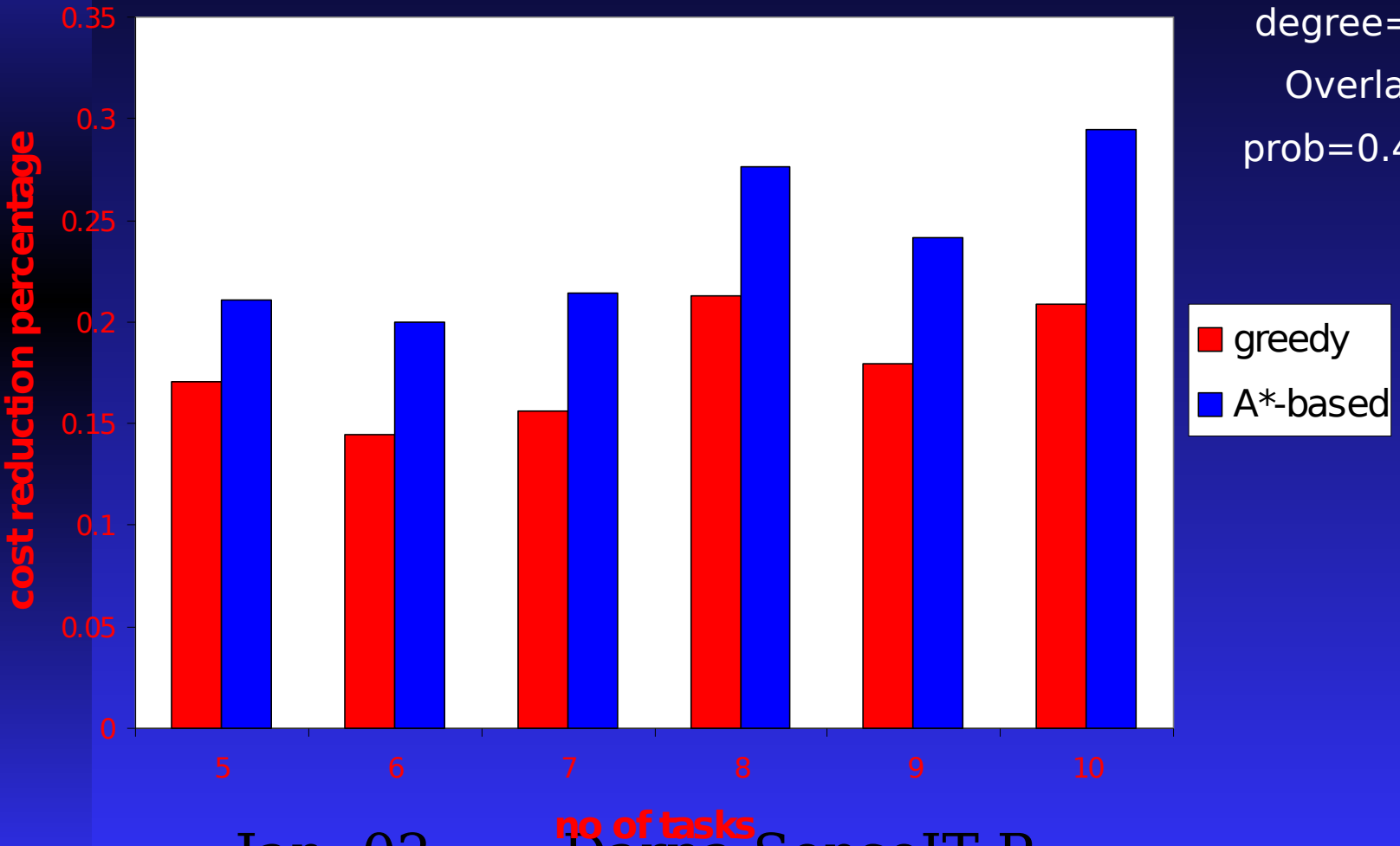


Cost Reduction

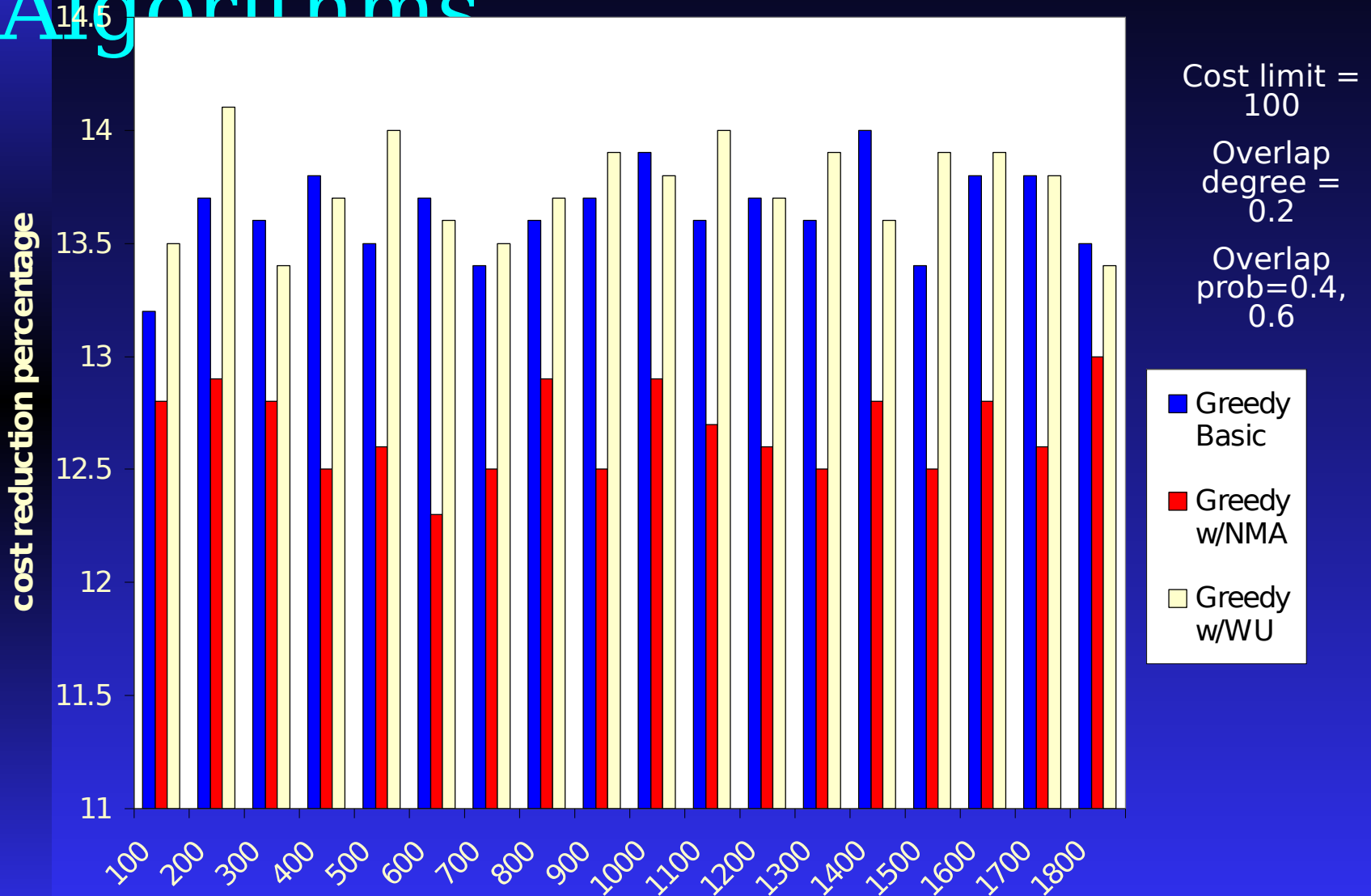
Cost-limit=100

Overlap-degree=0.6

Overlap-prob=0.4,0.6



Cost Reduction of Greedy Algorithms



Bottom Line

- Both A*-based and the BAB algorithm finds optimal solution, but do not scale
- Greedy algorithms find “good” solutions and scale up well
  - Greedy w/NMA scales very well, but achieves smaller cost reduction percentages
  - Greedy w/WU achieves very large cost savings; it becomes the clear winner as the overlap degree increases
- Partitioning algorithms, in conjunction with merging algorithms promise substantial scalability improvements.

Other key contributions

- Solved task assignment problem efficiently despite NP-completeness. (Golubchik,Ozcan, Subrahmanian).
- Temporal probabilistic relational DBs on top of ODBC (TODS 2001)
- Solved problem of scaling temporal probabilistic databases –Built cost models and query optimizer. (Dekhtyar, Ross, Ozcan, Subrahmanian)
- Probabilistic object base models (TODS 2001)
- Temporal probabilistic object base models (sub)

SenseIT group demos

- Developed gateway framework for communicating with on-node cache maintained by Fantastic data.
- STM provides data conduit behind Va Tech GUI.
- Participated in Nov. 2002 SITEX experiments at 29 Palms. UMD gateway and conduit used there.
- UMD Gateway and STM also to be used in joint demo with BBN, Va Tech, and other team members tomorrow.

Contact Info

- V.S. Subrahmanian
- Dept. of Computer Science
  AV Williams Building
  University of Maryland
  College Park,MD 20742.
- Tel: (301) 405-2711
- Fax: (301) 405-8488
- Email: vs@cs.umd.edu
- URL: www.cs.umd.edu/users/vs/index.html